González, S., Oliva, A., de la, Bernardos, C.J., Contreras, L.M. (2018). *Towards a Resilient Openflow Channel Through MPTCP*. Paper submitted in 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Valencia.

DOI:10.1109/BMSB.2018.8436865

Towards a resilient OpenFlow channel through MPTCP

Sergio González*, Antonio de la Oliva*, Carlos J. Bernardos*, Luis M. Contreras†
*Universidad Carlos III de Madrid, †Telefónica I+D

***Abstract*—In the recent years, Software Defined Networking (SDN) has changed the way networks are engineered, making them more flexible, programmable and dynamic. SDN advocates for the centralization of control functionalities in a central node, the so-called controller. This entity has a wide view of the entire network, including the topology, facilitating the management and decreasing the complexity. However, the existence of a single entity running the complete control plane constitutes a single point of failure, thus triggering the need of improving the resiliency and reliability of the controller and the connection between the control and the data plane.**

This paper presents a solution for the improvement of the resiliency and the reliability on the OpenFlow channel through the use of multipath TCP (MPTCP). The proposed solution is based on the simultaneous use of in-band and out-of-band paths for the OpenFlow control channel, and includes a first experimental evaluation of the performance gains that can be achieved.

***Index Terms*—SDN, OpenFlow, Resiliency, Reliability, MPTCP.**

I. Introduction and motivation

Current deployed networks are based on a distributed architecture with both control and data planes residing inside the routers and switches. This network design often suffers from vendor lock-in and configuration complexity, making the network complex, static, rigid and hard to manage.

As a consequence, current communication networks demand increasing flexibility, reconfigurability and lower cost of service deployment. Future network designs are focusing on the Software Defined Networking (SDN) paradigm as the architecture of choice to migrate the actual deployments [1] [2]. SDN approaches are based on decoupling the control and data planes while centralizing the network control logic on a node known as controller, moving towards an architecture that facilitates dynamic service creation and innovation.

In the recent years, SDN has focused the attention of academy and industry. This fact triggered the foundation of the Open Networking Foundation (ONF) [3] by Deutsche Telekom, Facebook, Google, Microsoft, Verizon and Yahoo with the objective of promoting SDN and standardize the OpenFlow [4] protocol. OpenFlow is a protocol defined between the controller and the data plane elements allowing the control and manipulation of the forwarding behavior of the network devices.

The control plane centralization enabled by SDN facilitates on the one hand the management of the network, but on the other hand it poses reliability, scalability and security issues. A centralized controller is a potential single point of failure and a potential bottleneck. A failure on the controller or a disconnection between the control and data planes may lead to performance degradation and packet loss.

OpenFlow uses a TCP connection (secured using SSL/TLS) to control the data plane elements. This connection, the so called OpenFlow channel, may use in-band or out-of-band transport signaling. In-band signaling is characterized by carrying the OpenFlow protocol packets through the same paths as the data transport, and therefore requires the network to be preconfigured to forward the OpenFlow signaling. Out-of-band signaling requires a separate network connecting all data plane elements to the controller, therefore requiring of extra network deployments.

There is a need to improve the resiliency and reliability on software defined networks. Some efforts are devoted on the controller side, supporting a control plane distribution, starting with the version 1.2 of the OpenFlow protocol which supports mechanisms to use several simultaneous controllers. This concept was extended by diverse projects to increase the reliability on the control plane increasing the scalability and avoiding the single point of failure of a centralized architecture:

- Onix [5] presents a platform for building a control plane on top of it as a distributed system. It provides an API which consists on a data model to abstract the network infrastructure, this API provides control logic, the possibility of read/write the state of the data plane and a notification engine for network state changes.
- HyperFlow [6] solution is based on a logically centralized but physically distributed control plane synchronized with a publish/subscription system. All the controllers using HyperFlow have a consistent network-wide view and they run as if they are controlling the whole network.
- ElastiCon [7] propose an elastic distributed controller architecture with a controller pool which can grow or shrunk dynamically depending on the load or performance requirements. To make it possible ElastiCon has implemented a 4-phase controller migration protocol to adjust the switch-controller mapping depending on the distributed controller load.
- Some controller frameworks, such as the Open Networking Operating System ONOS [8], provide mechanisms for clustering multiple controller nodes. In an ONOS cluster each node knows the state of a network subsection and disseminates this local state across the cluster in an event-

based procedure.

Another approach is to increase the reliability on the control channel reducing the probability of a disconnection due to a link failure. Our solution focuses on the latter mechanism, increasing the OpenFlow channel resiliency and reliability. Our approach is based on providing the OpenFlow channel with multiple alternative paths, using simultaneously several out-of-band and in-band connections. Our proposal does not only improve the reliability of the OpenFlow channel in general scenarios when both in-band and out-of-band paths are available, but also facilitates the deployment of SDN-controlled networks where out-of-band signaling is not cost-wise feasible (e.g., mmWave mesh network deployments on lamp posts).

The reminder of this paper is hence structured as follows: Section II introduces the resiliency improvement on the OpenFlow channel through multipath protocols; the core of the paper is composed of Section III and Section IV which detail the proposed solution and provides experimental validation end evaluation results. We enumerate different aspects which may be improved in Section V. Finally Section VI draws the conclusions of this paper and the future work.

II. Multipath TCP and OpenFlow

Multipath TCP (MPTCP) [9] is a set of extensions to regular TCP allowing a TCP connection to use multiple paths to maximize throughput and increase the redundancy. MPTCP is designed in such a way that all multipath operations are hidden from the application. The application opens a standard TCP socket, while the layer below takes care of opening multiple TCP subflows and manage the ordered delivery of the diverse TCP segments from the application. Hence, the application does not require any modification to use MPTCP. Fig. 1 depicts the MPTCP operation.

Each MPTCP connection is formed by different TCP subflows differentiated by IP address, port, or both. From the network point of view, each subflow behaves as a normal TCP flow. MPTCP is in charge of the subflow management and provides different algorithms to decide through which subflow a segment must be sent, in order to provide ordered delivery, each segment is numerated.

When OpenFlow is used jointly with MPTCP, the controller and the data plane elements are only aware of a single TCP flow between them, although in reality several TCP subflows are transported through the network.

The use of MPTCP and OpenFlow has been mainly analyzed in the context of the data plane:

- In [10] the authors present a prototype for a multipath network using MPTCP in the end hosts to distribute the traffic across multiple paths and OpenFlow in the data plane to do the wide area traffic engineering.
- The work detailed in [11] shows some experiences with MPTCP in an intercontinental OpenFlow testbed. The authors use OpenFlow to discover the topology of the network, calculate multiple paths and configure those paths on the OpenFlow network and then they use MPTCP on the servers to distribute the load across these paths.
- In [12] the authors establish a special purpose control and measurement framework on top of two large-scale internet testbeds (GÉANT and PlanetLab Europe) enabling measurements with real traffic.
- The authors of [13] propose a solution to avoid multipath bottlenecks developing SMOC, a simple multipath OpenFlow controller, that uses only topology information of the network to avoid the collisions.

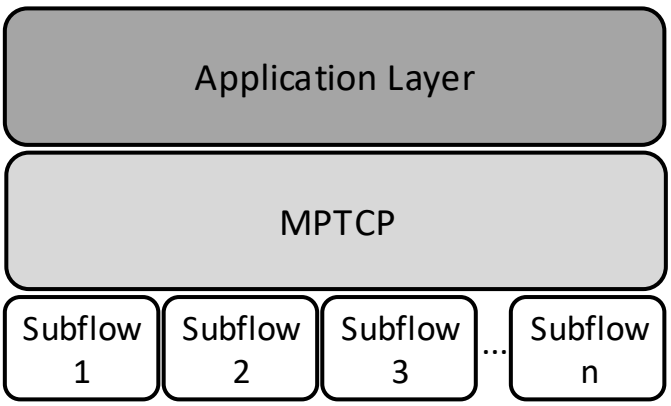


Fig. 1: MPTCP Operation

The previously mentioned works explore how to deal with the different TCP subflows in the data plane, in order to increase the performance and reliability of the end user flows.

The usage of MPTCP in the OpenFlow channel has been explored recently in [14], enhancing the performance using several out-of-band paths, e.g., connecting the controller through wireless and wired technologies to the data plane elements.

Our solution goes deeper into this concept providing mechanisms to enhance the reliability and the resiliency using multiple simultaneous in-band and out-of-band paths, securing the connection between the controller and the data plane elements providing alternative paths in case of link failure.

III. Proposed solution

Our solution is based on using MPTCP to allow the creation of multiple disjoint paths through the out-of-band and in-band management networks. The simultaneous use of these alternative paths increases both the robustness and the scalability of the OpenFlow channel and enables the reduction of the expenditure of deployment compared to other costly technologies (e.g., cellular).

To develop our solution we have used the Linux MPTCP Kernel implementation provided by IP Networking Lab at the Université Catholique de Louvain in Belgium [15]. This MPTCP Linux Kernel implementation has the following configurable parameters:

- The scheduler, it is in charge of distributing the segments across the different subflows, it can be managed in diverse ways according to these policies:

  - Default: the data is sent on the subflows with the lowest Round Trip Time (RTT) until the congestion window is full, then, the data is sent on the next subflow with lowest RTT.
  - Round-robin: the data is transmitted in a round-robin fashion trying to fill the congestion window on all subflows.
- The path-manager, it is in charge of the creation of the subflows, they can be created following these options:
  - Default: with this configuration the host does not announce IP addresses and does not create subflows but it accept the passive creation of subflows.
  - Fullmesh: with this option the host creates a full-mesh of subflows among each tuple $\{IP_{src}, IP_{dst}\}$ available.
  - Ndiffports: for a given tuple of IP addresses, MPTCP can create an arbitrary number of subflows using random TCP source ports.
- The congestion control of MPTCP, it can be managed in different manners:
  - Linked Increase Algorithm (LIA) [16], it couples the additive increase function of the different subflows with the resource pooling principle and uses the standard TCP behavior in case of drop. This algorithm allows to shift traffic from more congested paths to the less ones. The Linux Kernel implementation uses this algorithm by default.
  - Delay-based Algorithm or weighted Vegas (wVegas) [17], it adopts the packet queuing delay as the congestion signal, this approach is more sensitive to the changes of network congestion.
  - Opportunistic Linked-Increases Algorithm (OLIA) [18], like LIA it couples the additive increases and uses the standard TCP behavior in the case of drop, but differs in the increase part adapting the window increment as a function of the RTTs.
  - Balanced Linked Adaptation Algorithm (BALIA) [19], LIA and OLIA suffer from either unfriendliness to Single Path TCP (SPTCP) or unresponsiveness to network changes under certain conditions. BALIA generalizes the previously mentioned algorithms and strikes a good balance among TCP-friendliness, responsiveness, and window oscillation.

Our approach adopts the default scheduler, the fullmesh path-manager and the default congestion control. This configuration allows us to dynamically create more subflows by only adding an IP address. This dynamic procedure is not possible with ndiffports because the number of subflows must be predefined before the execution in the path-manager configuration [1].

In our setup, each IP address belongs to a different network, and for each data plane element we have included as many IP addresses reachable through the in-band signaling path as needed in function of the number of OpenFlow channel subflows. The reason for this can be found on the dynamic nature of our algorithm for path selection. We start the OpenFlow channel connection through an out-of-band signaling path. The out-of-band signaling does not require any flow configuration on the data plane elements because it is a dedicated channel with a direct connection to the controller. Through this channel the controller is able to learn the topology of the network. Based on that, the application at the controller decides the best in-band paths within the SDN-controlled network according to the number of desired redundant paths. Note that, at this point there is no in-band path created between the controller and the data plane elements, hence in-band communication is not possible.

Once the in-band paths have been selected, the controller sets up the paths within the SDN-controlled network creating one or several paths to each data plane element. This in-band path configuration is done through SDN flow tables matching by IP address. Then, the controller will be able to forward the OpenFlow messages through any of the in-band and out-of-band paths, moreover, the controller will be able to shut down any of the paths if it is desired, including the first path (out-of-band) which is the primary one for the TCP connection.

We have developed Algorithm 1 to compute the in-band paths. In this algorithm we first compute all the available paths between the targeted data plane element and the network controller through the in-band network, secondly we select the shortest path, then, depending on the desired redundancy we select the paths with less common links with the shortest, if more than one path have the same level of similarity we chose the shortest. This solution gives us the most disjoint-shortest paths of the in-band network.

Algorithm 1 Algorithm for OpenFlow channel path selection

```
1: shortestPath = Dijkstra(paths)
2: pathsSelected = [shortestPath]
3: while length(pathSelected) < numPaths do
4:     nextPaths = PathsLessCommonLinks(shortestPath)
5:     if nextPaths.length > 1 then
6:         selected = orderByNumHops(nextPaths).popleft()
7:     else
8:         selected = nextPaths
9:     pathsSelected.append(selected)
```

IV. Evaluation

We have created a simple scenario formed by a single controller and four data plane elements connected in a full-mesh topology, this scenario is depicted in Fig. 2. The network controller used in this testbed is Ryu [2], a component-based SDN framework written in Python. Ryu fully supports OpenFlow (v1.0 to v1.5) as well as Nicira Extension and Netconf. The main reason of choosing Ryu instead other controllers like ONOS or OpenDayLight was because it has good documentation, and we had previous experience developing applications in this SDN framework. Our Ryu controller runs in an Ubuntu

[1] https://multipath-tcp.org/pmwiki.php/Users/ConfigureMPTCP

[2] https://osrg.github.io/ryu/

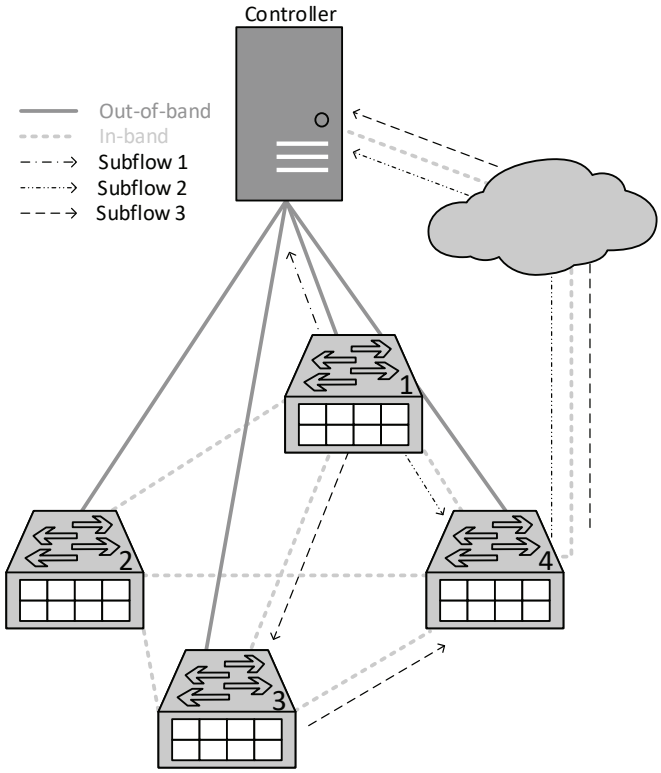


Fig. 2: Evaluation scenario

machine with an Intel Core i7 5557U@3.1GHz with 16GB of RAM and all the data plane elements are Alix2D3 devices running Debian and OpenVSwitch[3]. We have used the Linux MPTCP Kernel implementation v0.89 [15] on all the machines involved in the testbed. Each of the data plane elements has a direct connection to the controller (out-of band) and several connections through the data plane (in-band).

We propose the creation of multiple MPTCP subflows with disjoint paths (calculated with Algorithm 1) between each data plane element and the controller across both networks (out-of-band and in-band). In case the direct link between the switch and the controller through the out-of-band network fails, the OpenFlow channel connection is kept alive through the in-band network. The OpenFlow channel will remain active as long as at least one of the TCP subflows is alive.

In the experiments described in the following lines, we have generated OpenFlow traffic (packet-in messages) from the data plane elements to the controller with the objective of measuring the TCP subflow change. In order to generate periodic traffic on the OpenFlow channel we have used the linux `ping` tool configured to send ICMP messages each 1 ms, the data plane elements does't have a configured flow entry for ICMP messages hence each ICMP message will generate a packet-in to the network controller.

The first step is a validation test. As shown in Fig. 2, there are three different TCP subflows between the controller and data plane element 1, a primary out-of-band subflow and two secondary in-band subflows. The subflows across the in-band channel are elected firstly by length (shortest) and secondly by dis-junction with the shortest path. By default, the traffic travels through the primary TCP subflow.

At some point of time, we cause a link failure in the primary flow, MPTCP mechanisms detects the failure and moves the traffic to the secondary subflow, then we cause again a link failure in the secondary subflow forcing the traffic to move to the third subflow. Figure 3a shows the packets exchanged through the OpenFlow channel for each of the TCP subflows. As can be seen the disruption is minimal (we quantify the disruption in the next experiment) and packets continue flowing, hence the OpenFlow channel is not disrupted in any case.

The second experiment is focused on the performance evaluation of our solution. We measured the subflow handover delay depending on the number of hops towards the controller: directly connected, one hop and two hops distanced. This experiment is performed by setting one out-of-band and two in-band paths from data plane element 1 to the controller as shown in Fig. 2.

Then, we have measured the time needed to move the traffic to other subflow in three situations: when a link directly connected to a data plane element fails, when a one-hop distanced link fails and finally when the link two-hops distanced fails. The Cumulative Distribution Function (CDF) of the handover delay is presented in Fig. 3b. The average handover delay obtained is approximately $160ms$ for the directly connected link failure, $210ms$ for a one-hop-distanced link failure and $220ms$ for the two-hops distanced.

V. Improving MPTCP for resiliency of the control plane

The use of legacy MPTCP for the OpenFlow channel brings reliability advantages. Nevertheless, we have identified several aspects which may improve the behavior of MPTCP in scenarios such the one addressed in this paper. The main problem we have faced is the lack of control over the MPTCP operation from the applications on top of it, MPTCP does not allow the user to control in an easy way the operations of each subflow:

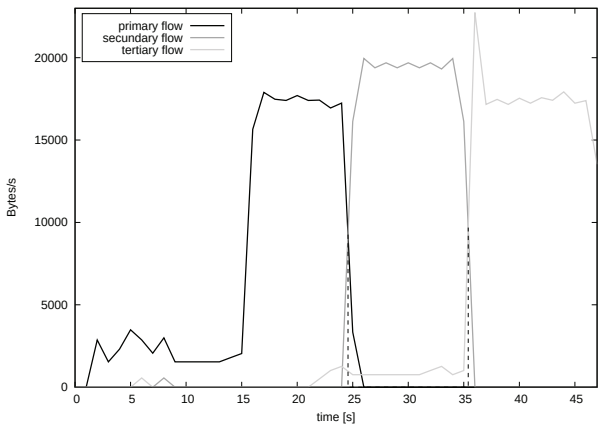
- If the IP address of the primary subflow is deleted all the subflows fails.
- If the primary subflow fails the traffic moves to a secondary subflow, if all the secondary subflows fail then the primary subflow is needed to recover the MPTCP connection.

Most of these issues can be solved by allowing the user to change the primary flow used for a given MPTCP connection.

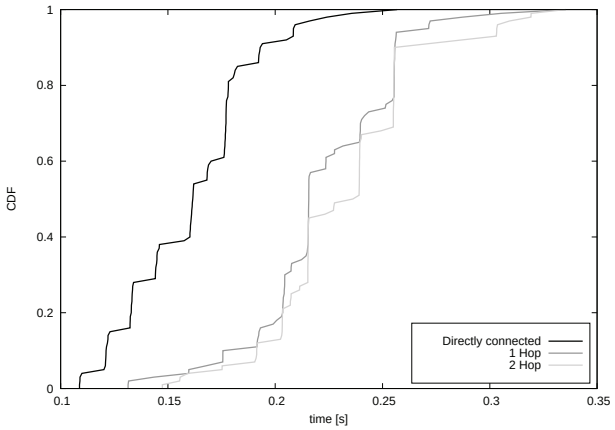
VI. Conclusions and Future Work

The evolution of current network solutions is clearly passing by the Software Defined Networking paradigm, but the concept of a centralized architecture incurs on some reliability and resiliency issues, e.g., the controller is a potential single

[3]http://openvswitch.org



(a) Validation test



(b) CDF of TCP subflow handover

Fig. 3: Results

point of failure, because of this, there have been an increasing effort on improving the reliability of the network controller and the OpenFlow channel.

In this paper we propose a solution for the OpenFlow channel based on the use of a multipath protocol (MPTCP) to provide alternative redundant paths through the in-band and out-of-band management networks. The selection algorithm of these disjoint-shortest paths of the in-band network is detailed in addition to the validation and evaluation of the presented solution. The results provided in this paper show that our proposal is resilient against path failure increasing both the robustness and the scalability of the OpenFlow channel.

The validation and evaluation was conducted on a real physical environment composed of five machines (four switches and a network controller). A physical scenario is limited by the available hardware and it is really difficult to have a massive scale setup, as future work we are working on more complex evaluation procedures using simulations or either emulated virtual networks with mininet [4]. A large scale scenario will allow us to generate more disjoint paths over the in-band network and will let us perform more complex tests and gather more information and results.

[4]https://github.com/mininet/mininet

ACKNOWLEDGEMENT

This work has been partially funded by the H2020 collaborative Europe/Taiwan research project 5G-CORAL (grant num. 761586) and the EU H2020 5G-TRANSFORMER Project (grant num. 761536).

REFERENCES

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[2] "CORD: Central Offices Re-architected as Datacenters." [Online]. Available: https://wiki.onosproject.org/pages/viewpage.action?pageId=3441030

[3] ONF, "Open Networking Foundation," https://www.opennetworking.org/, accessed: 2017-07-18.

[4] ONF, "OpenFlow switch specification," Open Networking Foundation, Version 1.5.0, Dec 2014.

[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A Distributed Control Platform for Large-scale Production Networks," in *OSDI*, vol. 10, 2010, pp. 1–6.

[6] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, pp. 3–3.

[7] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 7–12, 2013.

[8] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[9] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses. RFC6824, 2014."

[10] R. Van Der Pol, S. Boele, F. Dijkstra, A. Barczyk, G. van Malenstein, J. H. Chen, and J. Mambretti, "Multipathing with mptcp and openflow," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*. IEEE, 2012, pp. 1617–1624.

[11] R. van der Pol, M. Bredel, A. Barczyk, B. Overeinder, N. van Adrichem, and F. Kuipers, "Experiences with MPTCP in an intercontinental OpenFlow network."

[12] B. Sonkoly, F. Németh, L. Csikor, L. Gulyás, and A. Gulyás, "Sdn based testbeds for evaluating and promoting multipath tcp," in *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3044–3050.

[13] C. Nakasan, K. Ichikawa, H. Iida, and P. Uthayopas, "A simple multipath openflow controller using topology-based algorithm for multipath tcp," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 13, 2017.

[14] K. Nguyen, K. Ishizu, H. Murakami, F. Kojima, and H. Yano, "A Scalable and Robust OpenFlow Channel for Software Defined Wireless Access Networks," in *Vehicular Technology Conference (VTC Fall), 2015 IEEE 82nd*. IEEE, 2015, pp. 1–5.

[15] C. Paasch, S. Barré *et al.*, "Multipath TCP in the Linux kernel," www.multipath-tcp.org, accessed: 2017-07-18.

[16] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," Tech. Rep., 2011.

[17] M. Xu, Y. Cao, and E. Dong, "Delay-based congestion control for mptcp," *IETF, Individual Submission, Internet Draft draft-xu-mptcpcongestion-control-02*, 2015.

[18] R. Khalili, N. Gast, M. Popovic *et al.*, "Opportunistic linked-increases congestion control algorithm for mptcp," 2013.

[19] A. Walid, Q. Peng, J. Hwang, and S. H. Low, "Balanced linked adaptation congestion control algorithm for mptcp," *IETF, Individual Submission, Internet Draft draft-walid-mptcp-congestion-control-03*, 2015.